Тестовое задание

Техническая записка и скриншоты

Table of Content

[1 Repositories: 2](#_Toc153327945)

[2 Техническое задание: 2](#_Toc153327946)

[2.1 Дополнительные плюсы: 3](#_Toc153327947)

[3 Техническая записка 3](#_Toc153327948)

[3.1 Версии SW 3](#_Toc153327949)

[3.2 Варианты решения задачи 3](#_Toc153327950)

[3.3 Подключение БД 3](#_Toc153327951)

[3.4 Тестовые данные 4](#_Toc153327952)

[3.5 Реализованные механизмы 4](#_Toc153327953)

[3.6 Варианты запуска программы 5](#_Toc153327954)

[4 Скриншоты 6](#_Toc153327955)

[4.1 Развертывание приложения в Docker 6](#_Toc153327956)

[4.2 Работа приложения в Docker 7](#_Toc153327957)

[4.3 Ответ по запросу http://127.0.0.1:8000/?category=train&category=bus - Image8.jpg 8](#_Toc153327958)

[4.4 Следующий ответ по запросу http://127.0.0.1:8000/?category=train&category=bus - Image9.jpg 9](#_Toc153327959)

[4.5 Ответ по запросу если категория не найдена 10](#_Toc153327960)

# Repositories:

<https://github.com/ecrvmal/RTSoft> - for native implementation

<https://github.com/ecrvmal/RTSoft/tree/Docker_Implementation> - for docker implementation

# Техническое задание:

Задача - сделать сервис просмотра контента.

Он должен представлять из себя простой веб-сервис написанный на python 3 с использованием любых показавшихся необходимыми библиотек.

Есть не изменяемый файл конфигурации в CSV, который содержит данные вида:

Image\_URL;needed\_amount\_of\_shows;category1;category2;category3; … ;category N

Количество записей в файле не более 1000.

Файл содержит только символы.

Необходимое количество показов  - сколько раз картинку нужно показать.

Категорий у одной картинки может быть от 1 до 10 штук, записанных в соответствующих колонках.

Пример:

<http://localhost:8080/static/image1.jpg;500;flight;airlplane>

<http://localhost:8080/static/image2.jpg;3300;show;britain;bennyhill;sketches;tv>

<http://localhost:8080/static/image3.jpg;1500;games;minecraft;blocks;sandbox>

<http://localhost:8080/static/image4.jpg;120;onlycategory>

При запуске сервис считывает конфигурационный файл и начинает слушать HTTP обращения.

Клиент HTTP GET запросом сообщает категории, которые он готов принять, к примеру:

<http://localhost:8080/?category>[]=auto&category[]=trains

Количество запрошенных категорий может быть от 0 до 10 штук.

В ответ на него сервис должен выдавать простую HTML обертку с изображением картинки, которая совпадает минимум по одной категории.

Категории картинки произвольны если в запросе категорий не указано.

Каждый показ вычитается из количества необходимых к показу.

По характеру запросов считаем, что:

1. не будут запрашиваться категории, которых нет в конфигурационном файле.
2. количество запрошенных категорий равновероятно в рамках 0-10.
3. появление любой из категорий в запросе равновероятно.

## Дополнительные плюсы:

1. Код соответствует PEP8.
2. Механизм, который уменьшает вероятность выдачи одной и той же картинки несколько раз подряд
3. Механизм, позволяющий минимизировать вероятность возникновения случаев, когда подходящие картинки уже исчерпали свой лимит и ответить на запрос нечем.

# Техническая записка

## Версии SW

Для выплнения задачи была использована библиотека Django версии 3.2.23,   
Версия интерпретатора Python 3.10  
Версии используемых модулей приведены в репозитории в файле requirements.txt

## Варианты решения задачи

Поставленную задачу можно было выполнить 2-мя способами:

1. без создания БД, в этом случае файл CSV нужно подгружать каждый раз снова, результаты прошлой сессии не сохраняются, и
2. c созданием БД- в этом случае файл CSV загружается 1 раз командами

python manage import\_categs.py <filename.csv>

python manage import\_data.py <filename.csv>

и в дальнейшем код работает с данными, загруженными в БД. При перезагрузке программы данные не теряются.

В проекте реализован 2-й вариант.

## Подключение БД

В качестве БД в проекте используется Рostgres. Перед запуском монолитного варианта Django для работы с БД нужно обеспечить связь программы и БД, для этого сделать следующее:

* поправить строки в config/settings.py (в части user/password/name )

DATABASES = {

"default": {

"ENGINE": "django.db.backends.postgresql\_psycopg2",

"NAME": "RTSoft",

"USER": "prj\_user",

"PASSWORD": "prj\_user",

"HOST": "127.0.0.1",

"PORT": "5432",

}

}

* Также установить БД Postgres
* Cоздать в Postgres базу данных с названием "RTSoft".

## Тестовые данные

Для отладки программы были созданы тестовые изображения и тестовый CSV файл.

Данные созданы с помощью скрипта расположенного в директории /gen\_csv/gen\_csv.py

Тестовый файл SCV

image1.jpg;2106;bridge;station;shop;hotel  
image2.jpg;2677;pilot;ship;boat;museum;park;station;city;highway  
image3.jpg;2199;tower;ticket;pilot;street  
image4.jpg;2439;tower;park;highway;bridge;parking;hotel  
image5.jpg;132;station;bus;bridge;boat;ship  
image6.jpg;1536;bike;parking;boat;city;station;airport;train;highway  
image7.jpg;419;station;bike  
image8.jpg;1967;bus;bike;shop;city  
image9.jpg;1931;hotel;city;flight;street;train

Screenshot

Примеры тестовых изображений (jpg формат) :

Screenshot

Тестовые данные можно восстановить из фикстуры командой (Вместо загрузки данных командами) :

python manage.py loaddata mainapp/fixtures/001\_dumpdata.json

Тестовые изобрвжения загружены в фолдер /static/ , тестовый файл сохранен как /gen\_csv/input\_data.csv

## Реализованные механизмы

В процессе написания кода реализован механизм, который уменьшает вероятность выдачи одной и той же картинки несколько раз подряд. Это обеспечивается тем, что из списка изображений выбирается изображение с изображение с максимальным количеством оставшихся показов, если оно совпадает с ранее показанным – выбирается следующее изображение из списка, отсортированного по количеству показов.

Также реализован механизм, позволяющий минимизировать вероятность возникновения случаев, когда подходящие картинки уже исчерпали свой лимит и ответить на запрос нечем : в начале работы программы из набора возможных изображений выбирается изображение с максимальным количеством оставшихся показов, при выравнивании оставшегося количества показов картинки показываются равномерно.

Реализован контроль количества показа изображений, при исчерпании лимита показов – показывается изображение «по умолчанию» и выводится надпись   
« Items not found, Dummy image shown »

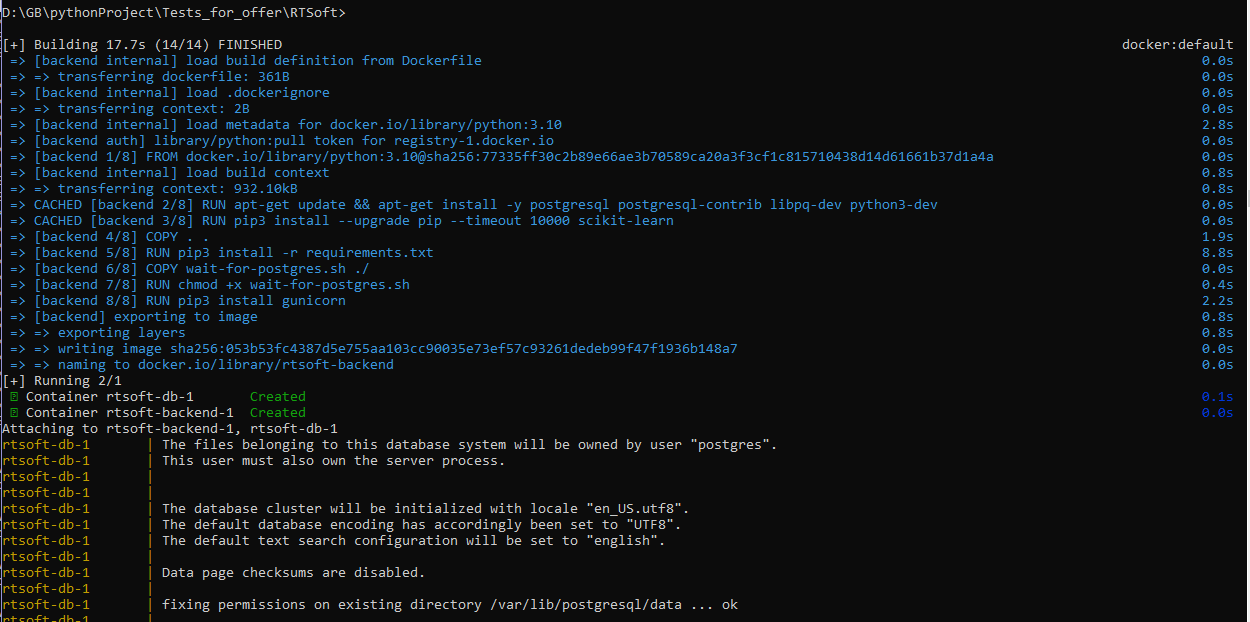
## Варианты запуска программы

Возможен запуск программы как в монолитном варианте, так и в среде Docker. Соответствующие файлы и настройки загружены в репозиторий.

При запуске в среде Docker для ускорения разработки был использован втроенный   
wsgi сервер, для коммерческого использования необходимо доработать программу, использовать сервер Gunicorn и сервер NGINX frontend для раздачи статики.

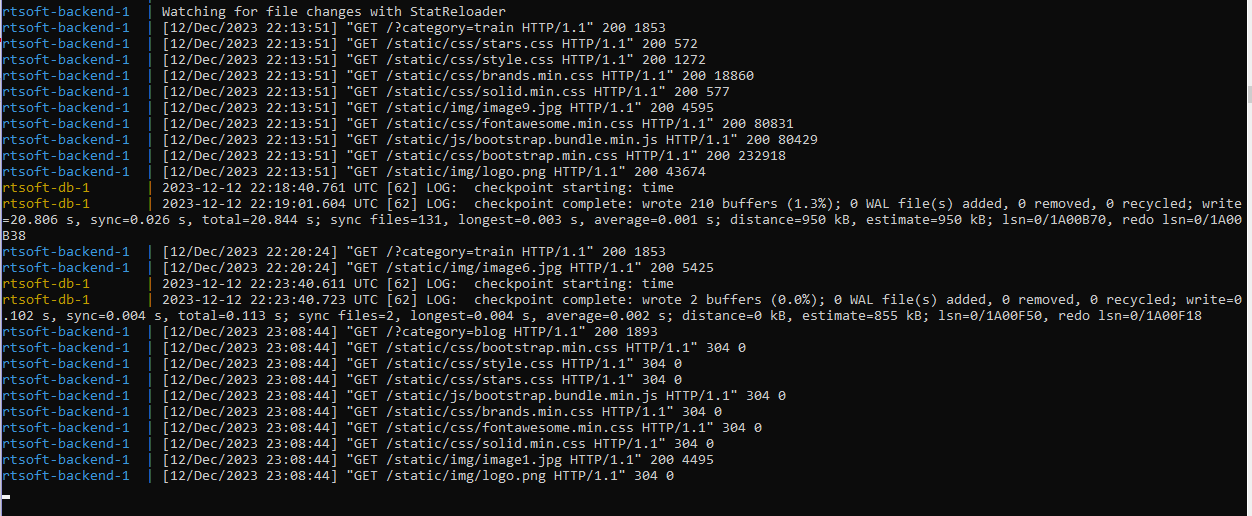
# Скриншоты

## Развертывание приложения в Docker



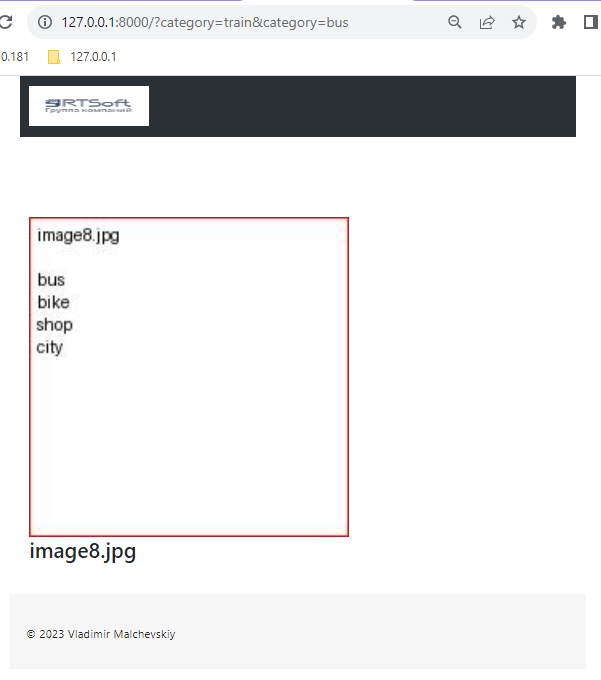
Screenshot

## Работа приложения в Docker



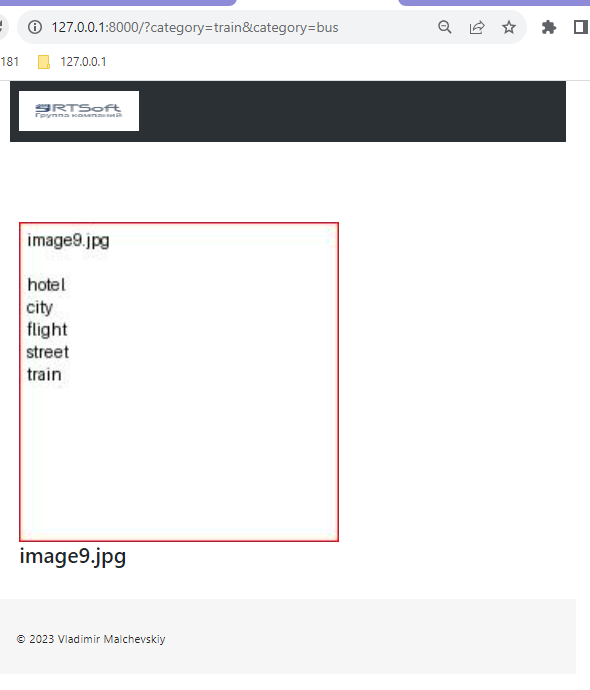
Screenshot

## Ответ по запросу <http://127.0.0.1:8000/?category=train&category=bus> - Image8.jpg



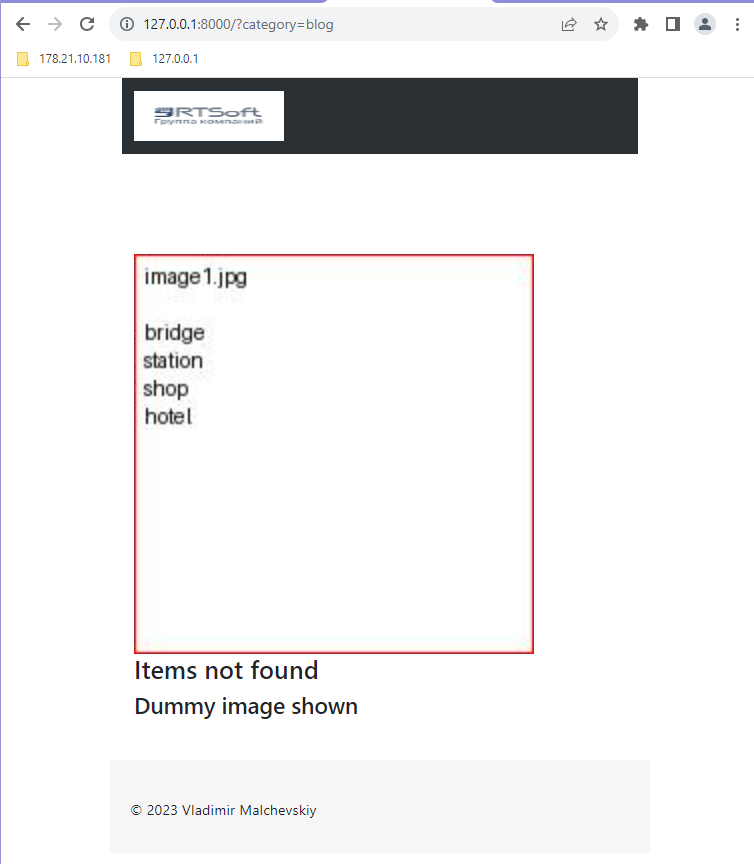
Screenshot

## Следующий ответ по запросу <http://127.0.0.1:8000/?category=train&category=bus> - Image9.jpg



Screenshot

## Ответ по запросу если категория не найдена



Screenshot